Interfaces in Java

In Java, an **interface** is a blueprint of a class that contains only **abstract methods** (methods without a body) and **constants** (variables declared as `public`, `static`, and `final` by default). Interfaces help achieve **abstraction** and **multiple inheritance**, making Java programs more modular and maintainable.

Why Use Interfaces?

- **Achieve Abstraction:** Interfaces allow us to define methods without implementing them, leaving the implementation to child classes.
- **Support Multiple Inheritance:** Unlike classes, Java allows a class to implement multiple interfaces, overcoming the limitations of single inheritance.
- **Ensure Loose Coupling:** Interfaces separate the definition of functionality from implementation, making code more flexible and scalable.

Defining and Implementing an Interface

Example: Defining an Interface

```java
interface Animal {
    void makeSound(); // Abstract method (no body)
}
```

Example: Implementing an Interface in a Class

```java
class Dog implements Animal {
    @Override
    public void makeSound() {
        System.out.println("Dog barks");
    }
}

public class InterfaceExample {
    public static void main(String[] args) {
        Animal myDog = new Dog(); // Upcasting
        myDog.makeSound();
    }
}
```

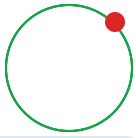
Output:

```
Dog barks
```

Here, `Dog` implements the `Animal` interface by providing a concrete definition for the `makeSound()` method.

Multiple Interfaces in Java

A class can implement multiple interfaces, which is not possible with regular class inheritance.



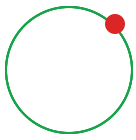
Example: Implementing Multiple Interfaces

```java
interface Flyable {
    void fly();
}

interface Swimmable {
    void swim();
}

class Duck implements Flyable, Swimmable {
    @Override
    public void fly() {
        System.out.println("Duck is flying");
    }

    @Override
    public void swim() {
        System.out.println("Duck is swimming");
    }
}

public class MultipleInterfacesExample {
    public static void main(String[] args) {
        Duck myDuck = new Duck();
        myDuck.fly();
        myDuck.swim();
    }
}
```

Output:

```
Duck is flying
Duck is swimming
```

This example shows how a class (`Duck`) can implement multiple interfaces (`Flyable` and `Swimmable`) and provide implementations for both.

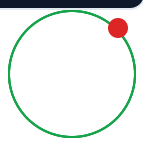


Default and Static Methods in Interfaces

Since Java 8, interfaces can have **default** and **static** methods with concrete implementations.

Example of Default and Static Methods

```
interface Vehicle {
    void start(); // Abstract method

    // Default method with a body
    default void stop() {
        System.out.println("Vehicle is stopping");
    }

    // Static method with a body
    static void maintenance() {
        System.out.println("Vehicle requires maintenance");
    }
}

class Car implements Vehicle {
    @Override
    public void start() {
        System.out.println("Car is starting");
    }
}

public class InterfaceMethodsExample {
    public static void main(String[] args) {
        Car myCar = new Car();
        myCar.start(); // Calls overridden method
        myCar.stop();  // Calls default method
        Vehicle.maintenance(); // Calls static method
    }
}
```



Output:

```
Car is starting
Vehicle is stopping
Vehicle requires maintenance
```

- **Default methods** allow interfaces to have some behavior without breaking existing implementations.
- **Static methods** belong to the interface itself and can be called without an instance.

Abstract Classes vs Interfaces

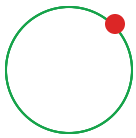
Feature	Abstract Class	Interface
Can have constructors?	✅ Yes	❌ No
Can have abstract methods?	✅ Yes	✅ Yes
Can have concrete methods?	✅ Yes	✅ (Since Java 8)
Can have instance variables?	✅ Yes	❌ No (Only constants allowed)
Supports multiple inheritance?	❌ No	✅ Yes

Do’s and Don’ts of Interfaces

✅ Do’s:

✅ **Use interfaces for abstraction**: If you only need to define behavior without implementing it, use an interface instead of an abstract class.

✅ **Use interfaces for multiple inheritance**: If a class needs behavior from multiple sources, interfaces help avoid the **diamond problem** found in multiple inheritance.



✅ **Use default methods to provide optional implementations**: If an interface needs to add new behavior without breaking existing classes, default methods help maintain backward compatibility.

❌ Don’ts:

❌ **Don't use interfaces for code reuse**: Interfaces only define behavior but don’t provide a way to share reusable logic across multiple classes. Use abstract classes if code sharing is needed.

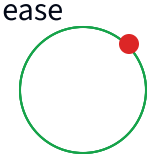
❌ **Don't create interfaces for single-use cases**: If an interface is implemented by only one class and is unlikely to be reused, a regular class or abstract class is a better choice.

❌ **Don't overuse default methods**: Default methods should be used sparingly, as too many can make the interface behave like an abstract class, defeating the purpose of interfaces.

Conclusion

In this blog, we learned about **Interfaces in Java**, how they allow **abstraction** and **multiple inheritance**, and when to use them. We explored:

- How to define and implement interfaces
- The concept of multiple interfaces
- Default and static methods in interfaces
- Best practices with **Do’s and Don’ts**



Last updated on ***Apr 09, 2025***

Subscribe to our newsletter

Read articles from Coding Shuttle directly inside your inbox. Subscribe to the newsletter, and don't miss out.

Enter your email address SUBSCRIBE

We recommend

Spring Boot 0 to 100 Cohort 4.0 [AI + DevOps]

₹9,990 ~~₹ 27,990~~

CHECK IT OUT →

Previous
« Abstract Class in Java

Next
Inner Classes in Java »

POPULAR HANDBOOKS

Spring Boot Handbook | Java Programming Handbook | Kubernetes Handbook

ONLINE COMPILERS

Online Java Compiler | Online C++ Compiler | Online Python Compiler |
Online Javascript Compiler | Online Go Compiler | Online C# Compiler | Online C Compiler

DSA SHEETS

CS 45 Sheet | CS SDE Sheet | DSA Prime Sheet | Blind 75 Sheet

MOCK TESTS

Quantitative Aptitude for Placements | Logical Aptitude for Placements | C++ OOPS for Interviews |
Java OOPS for Interviews | Javascript for Interviews | Python for Interviews |
C++ Language for Interviews | C Language for Interviews | Java for Interviews |
Operating Systems | SQL for Interviews | RDBMS for Interviews |
Docker & Kubernetes for Interviews | Spring Cloud for Interviews |
Advanced Spring Boot for Interviews | Spring Security for Interviews |